MINISTÉRIO DA DEFESA
EXÉRCITO BRASILEIRO
DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA
INSTITUTO MILITAR DE ENGENHARIA
CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

ANA CAROLINA GARCIA GUIMARÃES

LETÍCIA TIVERON BORGES TANNÚS

IMPLEMENTAÇÃO DE UM SISTEMA DE ESTEGANOGRAFIA COM CRIPTOGRAFIA VISUAL

Rio de Janeiro

2013

INSTITUTO MILITAR DE ENGENHARIA

ANA CAROLINA GARCIA GUIMARÃES

LETÍCIA TIVERON BORGES TANNÚS

IMPLEMENTAÇÃO DE UM SISTEMA DE ESTEGANOGRAFIA COM CRIPTOGRAFIA VISUAL

Projeto de Fim de Curso apresentado ao Curso de Graduação em Engenharia de Computação do Instituto Militar de Engenharia.

Orientador**:** José Antônio Moreira Xexéo - D. Sc.

Rio de Janeiro

2013

INSTITUTO MILITAR DE ENGENHARIA
Praça General Tibúrcio, 80 – Praia Vermelha
Rio de Janeiro – RJ CEP: 22290-270

Este exemplar é de propriedade do Instituto Militar de Engenharia, que poderá incluí-lo em base de dados, armazenar em computador, microfilmar ou adotar qualquer forma de arquivamento.

São permitidas a menção, reprodução parcial ou integral e a transmissão entre bibliotecas deste trabalho, sem modificação de seu texto, em qualquer meio que esteja ou venha a ser fixado, para pesquisa acadêmica, comentários e citações, desde que sem finalidade comercial e que seja feita a referência bibliográfica completa.

Os conceitos expressos neste trabalho são de responsabilidade dos autores e do orientador.

621.39 Guimarães, Ana Carolina Garcia.

G963i Implementação de um sistema de esteganografia com criptografia visual/ Ana Carolina Garcia Guimarães, Letícia Tiveron Borges Tannús; orientado por José Antônio Moreira Xexéo. -Rio de Janeiro: Instituto Militar de Engenharia, 2013.

60f: il.

Projeto de Final de Curso. - Instituto Militar de Engenharia. - Rio de Janeiro, 2013.

1. Engenharia de Computação. 2. Criptografia Visual. 3. Esteganografia. I. Tannús, Letícia Tiveron Borges. II. Xexéo, José Antônio Moreira. III. Titulo. IV. Instituto Militar de Engenharia.

CDD 621.39

INSTITUTO MILITAR DE ENGENHARIA

ANA CAROLINA GARCIA GUIMARÃES
LETÍCIA TIVERON BORGES TANNÚS

IMPLEMENTAÇÃO DE UM SISTEMA DE ESTEGANOGRAFIA COM CRIPTOGRAFIA VISUAL

Projeto de Fim de Curso apresentado ao Curso de Graduação em Engenharia de Computação do Instituto Militar de Engenharia.

Orientador: José Antônio Moreira Xexeo - D. Sc.

Aprovada em 11 de junho de 2013 pela seguinte Banca Examinadora:

_____

Prof. José Antônio Moreira Xexeo - D. Sc., do IME

_____

Prof. Ricardo Choren Noya - D. Sc., do IME

_____

Prof. Paulo Roberto Gomes - M. Sc., do IME

Rio de Janeiro
2013

SUMÁRIO

RESUMO

Em 1994, Adi Shamir e Moni Naor apresentaram um novo modelo de criptografia seguro e fácil de usar, a criptografia visual. Neste trabalho é implementado um sistema criptográfico de esteganografia combinado com criptografia visual baseado em um sistema de marca d'água. Diferentemente dos outros trabalhos pesquisados, que aplicam criptografia visual e posteriormente a esteganografia, esse trabalho propõe o uso das duas técnicas simultaneamente, de tal modo que a imagem que "esconderia" a mensagem pode ser interpretada como uma das transparências do modelo de Naor e Shamir.

Palavras-chave: criptografia, visual, esteganografia.

ABSTRACT

In 1994, Moni Naor and Adi Shamir presented a new model of encryption safe and easy to use, visual cryptography. In this paper it is implemented a cryptographic system of steganography with visual cryptography based on a watermark system. Differently from the others papers studied, which apply visual cryptography first and then the steganography, this one proposes using both methods simultaneously so that the image which would “hide” the message can be seen as one of the shares of Naor & Shamir’s model.

Keywords: cryptography, visual, steganography.

1 INTRODUÇÃO

O interesse pelos assuntos aplicáveis à segurança das informações cresceu aceleradamente nos últimos anos. A globalização dos negócios e a exposição gerada pela Internet tornaram as grandes corporações civis, forças armadas e governo particularmente envolvidos na preservação do sigilo de seus arquivos e comunicações.

Entre os instrumentos de segurança está a criptografia, conjunto de conceitos e técnicas usados para codificar conteúdos de mensagens e torná-los incompreensíveis para pessoas não autorizadas. Os sistemas criptográficos compõem-se de remetente, mensagem, algoritmo e chave de cifrar, mensagem criptografada, algoritmo e chave de decifrar e destinatário, como representado na figura FIG. 1.1.

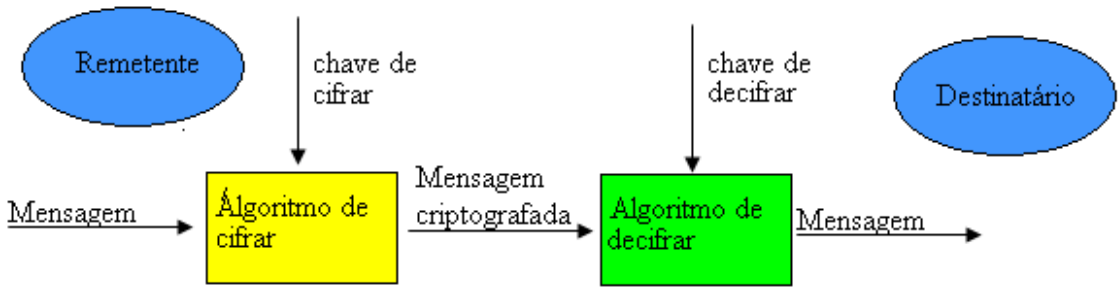


FIG. 1.1 - Modelo genérico de um sistema criptográfico

Supõe-se que os intrusos conhecem tudo, exceto as chaves. A criptoanálise, por outro lado, está a serviço do intruso disposto a "quebrar o sistema", ou para ler conteúdos ou para identificar a chave utilizada no processo de cifragem.

As técnicas de criptoanálise também se prestam para testar características de algoritmos criptográficos, na tentativa de melhorá-los. Atualmente, o grupo de pesquisa em Segurança da Informação do IME trabalha, principalmente, para obter, implementar e validar algoritmos criptográficos e, simultaneamente, prospectar novas técnicas aplicáveis à proteção e busca de informações sensíveis e análise de algoritmos.

Em algumas aplicações em que é necessário o uso de criptografia, pode não haver recursos disponíveis para realizar processamento computacional. Em 1994, Adi Shamir e Moni Naor apresentaram um novo modelo de criptografia, seguro e fácil de implementar, a criptografia visual.

Este modelo permite codificar materiais impressos como imagens, documentos e até textos escritos à mão. O processo de decodificação é realizado diretamente pelo sistema visual humano, e por isso independe de qualquer tipo de processamento criptográfico computacional. O modelo consiste na distribuição de k transparências entre n pessoas de modo que seja necessário que $k < n$ delas reúnam suas transparências para revelar a mensagem original.

1.1 OBJETIVO

O objetivo deste trabalho é implementar um sistema criptográfico real de esteganografia utilizando criptografia visual.

1.2 JUSTIFICATIVA

A criptografia é a arte de modificar uma mensagem a fim de que ela seja inteligível apenas para aqueles que conhecem seu protocolo de cifragem. Seu desenvolvimento se deve à necessidade dos governos manterem suas comunicações em sigilo. Nesse contexto a construção e a difusão desse conhecimento e a formação de pesquisadores nessa área podem ser de interesse militar.

A criptografia visual aplicada à esteganografia é uma técnica recente e que pode ser de interesse do Exército Brasileiro. O proposto por esse trabalho não foi encontrado em nenhuma outra bibliografia pelas autoras.

1.3 METODOLOGIA

Primeiramente, foram estudados sistemas e conceitos básicos de criptografia visual. Como referências iniciais, utilizaram-se (ANURADHA, 2010), (CHEN et al,2007) e (NAOR & SHAMIR, 1994). Também foi pesquisada uma aplicação real, o uso de criptografia visual em marcas d´água, que se relaciona com o sistema proposto.

Em seguida foi proposto um sistema para ser modelado e implementado neste trabalho, usando esteganografia com criptografia visual. Foram pesquisadas ferramentas pertinentes ao sistema proposto, além de outros trabalhos sobre o assunto, porém nenhum deles é semelhante a este trabalho.

O sistema básico, com duas transparências, de (NAOR & SHAMIR, 1994) foi implementado como teste para as ferramentas escolhidas e, em seguida, foi implementado o sistema proposto. Por fim, foi feito um estudo sobre os tipos de imagens mais adequados para o sistema e foi realizada uma análise de segurança para sua validação.

1.4 ESTRUTURA

O trabalho está estruturado da seguinte forma: No Capítulo 2 é feito um estudo sobre técnicas de criptografia visual. No Capítulo 3 é mostrada uma aplicação real que faz uso de técnicas de criptografia visual em sistemas de marca d'agua. O Capítulo 4 trata do uso da criptografia visual junto da esteganografia em outros trabalhos. No Capítulo 5 se encontra a proposta do sistema criptográfico implementado neste trabalho, bem como a especificação do algoritmo proposto e as ferramentas utilizadas. O Capítulo 6 mostra uma implementação inicial usando classes de manipulação de imagens em *C#* bem como a implementação do sistema proposto. Em seguida, no Capítulo 7, é exposto um estudo sobre a escolha das imagens para o sistema e no Capítulo 8 é feita a análise da segurança do mesmo. No Capítulo 9 encontra-se conclusão e a bibliografia está disponível no Capítulo 10. O Capítulo 11 contém o Apêndice com o código em *C#* das classes implementadas.

2 CRIPTOGRAFIA VISUAL

A Criptografia Visual foi introduzida por Naor e Shamir em 1994 (NAOR & SHAMIR, 1994) como uma solução alternativa ao problema de compartilhamento de segredos *k out of n*: Como dividir um segredo em *n* partes de forma que somente com $k<n$ delas seja possível recuperá-lo.

A proposta desse esquema é criar um sistema simples e seguro de criptografia de imagens, cuja decodificação possa ser feita sem auxílio computacional e sem necessidade de conhecimentos em criptologia.

Há também outras formas de se compartilhar segredos em Criptografia Visual. Neste Capítulo serão abordadas duas formas: o modelo de Naor e Shamir e um outro, chamado Random Grids.

2.1 O MODELO DE NAOR E SHAMIR

O modelo de Naor e Shamir tem por objetivo compartilhar uma imagem, que seria o segredo, e dividí-la em *n* partes, chamadas transparências, de forma que somente com $k<n$ transparências seja possível recuperar a imagem original.

Este modelo considera que a mensagem (imagem) é uma coleção de pixels brancos e pretos. Cada pixel é tratado separadamente e aparecerá de uma forma diferente em cada uma nas *n* transparências geradas. Cada transparência é uma coleção de *m* subpixels pretos e brancos, os quais são impressos muito próximos de forma que a visão humana faça uma média das contribuições individuais de cada pixel preto e branco. A estrutura construída é uma matriz *n* x *m* booleana $S = [s_{ij}]$, em que $s_{ij} = 1$ se, e somente se, o *j*-ésimo subpixel da *i*-ésima transparência é preto. Quando *k* transparências são sobrepostas de forma a alinhar os

subpixels correspondentes, forma-se uma imagem cujos subpixels pretos são representados pela operação "*or*" das linhas correspondentes às *k* transparências na matriz *S.* Conjuntos de subpixels muito próximos pretos e brancos são interpretados como cinza pelo olho humano, com o nível de cinza variando de acordo com a proporção dos pixels.

A solução de Naor e Shamir para compartilhamento de segredos *k out of n* consiste em dois conjuntos C_0 e C_1 de matrizes booleanas. Para compartilhar um pixel branco escolhe-se, aleatoriamente, uma das matrizes de C_0; para compartilhar um pixel preto escolhe-se, aleatoriamente, uma das matrizes de C_1. A matriz escolhida define a cor de *m* subpixels em cada uma das *n* transparências.

Para qualquer subconjunto $\{i_1, i_2,... i_q\}$ de $\{1,2,...n\}$ com $q < k$, as duas coleções de matrizes $q \times m$ D_t para $t \in \{0,1\}$ obtidas restringindo cada matriz $n \times m$ de C_t (com $t \in \{0,1\}$) às linhas $i_1, i_2,... i_q$, são indistinguíveis. Esta condição implica que não é possível decidir se cada pixel é branco ou preto unindo menos de *k* transparências, ou seja, não é possível reconstruir o segredo com menos de k transparências.

2.1.1 ALGORITMO 2 *OUT OF* 2 DE NAOR E SHAMIR

O problema de dividir um segredo em duas transparências, de forma que somente com ambas fosse possível recuperar o segredo, também foi descrito em (NAOR & SHAMIR, 1994).

É possível resolvê-lo mapeando cada pixel da imagem original em dois pixels em cada transparência, como indicado na tabela TAB. 2.1. Entretanto a imagem ficaria distorcida, "esticada", e por isso é mais adequado transformar cada pixel em um conjunto de quatro pixels em cada transparência. A correspondência é mostrada na tabela TAB. 2.2. Ao sobrepor as transparências, é feita uma operação *"or"* pixel a pixel, ou seja, a sobreposição de um pixel preto com qualquer outro pixel resulta em um pixel preto e a sobreposição de dois pixels brancos retorna um pixel branco.

TAB. 2.1 - Mapeamento em 2 sub-pixels

Pixel Original	Probabilidade	Sub-pixel 1	Sub-pixel 2	Sub-pixel 1 || Sub-pixel 2
	0,5			
	0,5			
	0,5			
	0,5			

TAB. 2.2 - Mapeamento em 4 sub-pixels

Pixel Original	Probabilidade	Sub-pixel 1	Sub-pixel 2	Sub-pixel 1 || Sub-pixel 2
	0,5			
	0,5			
	0,5			
	0,5			

No caso da tabela TAB. 2.2 a imagem criptografada terá, portanto, quatro vezes o tamanho da imagem original, mas manterá suas proporções. Nota-se, também, que um pixel branco da imagem original se transforma em um conjunto de dois pixels brancos e dois pretos na imagem criptografada. Ao olho humano, esse contraste de pixels próximos dá a sensação de cinza tornando a imagem criptografada compreendida. Esse ruído incluído na imagem criptografada garante a segurança do algoritmo, pois de posse de apenas uma transparência é impossível inferir da análise de um conjunto de pixels o conjunto correspondente na imagem em claro.

2.2 RANDOM GRIDS

O algoritmo de Naor e Shamir (NAOR & SHAMIR, 1994) provê segurança e é um sistema eficaz, porém a necessidade de expansão do pixel gera tempo extra de processamento e transição. Como uma alternativa a essa abordagem existem os algoritmos baseados em Random Grids.

O algoritmo consiste em dividir uma imagem *I* em duas transparências T_1 *e* T_2 com as mesmas dimensões de *I*. T_1 é uma coleção de bits pretos e brancos escolhidos aleatoriamente, enquanto T_2 é construída a partir de T_1 e da imagem *I*. Se um pixel de coordenadas (*i*, *j*) é branco em *I*, o pixel correspondente em T_2 será igual ao pixel de mesma coordenada de T_1. Caso o pixel de (*i*, *j*) seja preto, os pixels correspondentes de T_1 e T_2 devem ser complementares (KAFRI & KEREN, 1987). De forma geral, os pixels de T_1 *e* T_2 são escolhidos conforme indicado na tabela TAB. 2.3.

TAB. 2.3 - Escolha dos pixels de T_2 de acordo com os pixels em *I* e T_1

Pixel em *I*	Pixel em T_1	Probabilidade	Pixel em T_2	Pixel em T_1 || Pixel em T_2
□	□	0,5	□	□
□	■	0,5	■	■
■	□	0,5	■	■
■	■	0,5	□	■

Nota-se que um pixel preto na imagem original *I* continuará preto ao se unir T_1 *e* T_2, entretanto, um pixel branco em *I* ficará preto em 50% das vezes. Isso pode gerar uma diferença muito grande entre *I* e a imagem gerada por T_1 e T_2 quando *I* possuir mais de 50% de pixels brancos.

2.3 EXTENSÕES

2.3.1 CRIPTOGRAFIA VISUAL COM TRANSPARÊNCIAS CIRCULARES

Em (CHEN et al, 2007) é definido um esquema de criptografia visual que cria transparências circulares, ao invés de gerar transparências lineares como é feito no método de Naor e Shamir (NAOR & SHAMIR, 1994). Um conjunto de $n \geq 2$ imagens secretas é codificado em duas transparências circulares de forma que as imagens secretas podem ser recuperadas sobrepondo-se as duas transparências e rotacionando-se uma delas em n ângulos diferentes. Então, com o uso de apenas duas transparências, podem-se compartilhar mais de uma imagem.

O exemplo usado para descrever o sistema em (CHEN et al, 2007) usa $n = 3$ imagens secretas. Sejam:

• P_1, P_2 *e* P_3 as três imagens binárias secretas com as mesmas dimensões $h \times w$;

• p_1, p_2 *e* p_3 os pixels correspondentes nas imagens secretas P_1, P_2 *e* P_3, respectivamente;

• A e B as duas transparências circulares codificadas pelo esquema;

• $A \otimes B$ a sobreposição das transparências circulares A e B;

• $A^{\theta} \otimes B$ a sobreposição das transparências circulares A e B, com rotação da transparência A em θ graus no sentido horário.

O objetivo do sistema é garantir que $A \otimes B$ recupere P_1, $A^{120} \otimes B$ recupere P_2 e $A^{240} \otimes B$ recupere P_3, como mostra a figura FIG. 2.1.

Em (DOMINGUES, 2013) a criptografia com transparências circulares é usada para criar um novo esquema de prevenção contra fraudes dentro de um sistema *2 out of n* em que *n-1* participantes desonestos tentam enganar o *n*-ésimo participante honesto. Este novo esquema fornece a qualquer participante a capacidade de detectar a fraude, caso suspeite que não são

originais as transparências dos outros participantes ou a imagem decodificada após o processo de sobreposição.

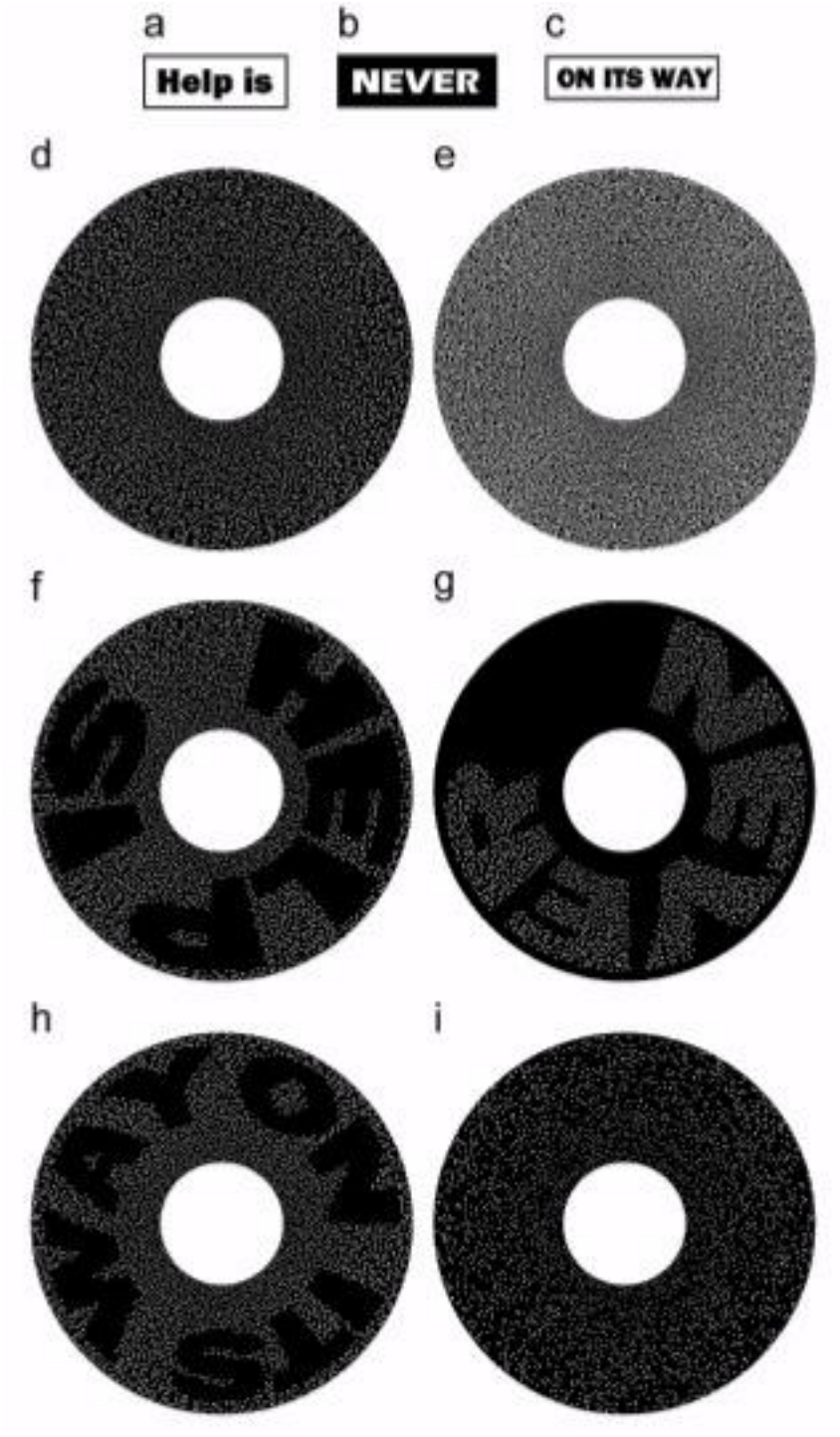


FIG. 2.1 - Resultados de uma implementação do compartilhamento de 03 imagens: (a) P_1, (b) P_2, (c) P_3, (d) *A*, (e) *B*, (f) A⊗B, (g) A^{120° ⊗B, (h) A^{240° ⊗B, (i) A^{85° ⊗B. (Extraído de (CHEN et al, 2007))

3 APLICAÇÃO EM SISTEMAS DE MARCA D'ÁGUA

Em (HWANG, 2000) é proposto um método de marca d'água baseado em criptografia visual segundo o qual o padrão de marca não é incorporado à imagem legítima diretamente, o que torna mais difícil detectá-la ou retirá-la, de forma ilegal, da imagem marcada . Ela pode ser recuperada a partir da imagem marcada sem uso de computação. O padrão de marca d'água pode ser qualquer imagem preto/branco significativa que caracterize o proprietário. Pelo método proposto, todos os pixels da imagem marcada são iguais aos da imagem legítima. Os resultados experimentais mostraram que o padrão de marca d'água na imagem marcada tem boa transparência, pois a marca não pode ser percebida, e boa robustez, já que, como não altera a imagem, não pode ser removida. (HWANG, 2000)

Esse é um problema *2 out of 2* de compartilhamento, ou seja, o segredo é dividido em 2 transparências e somente com a sobreposição de ambas o segredo é revelado. Nesse sistema, a marca d'água é o segredo e a imagem marcada é uma das transparências. A segunda transparência é gerada a partir da imagem, da marca e de uma chave secreta como mostra a figura FIG. 3.1.

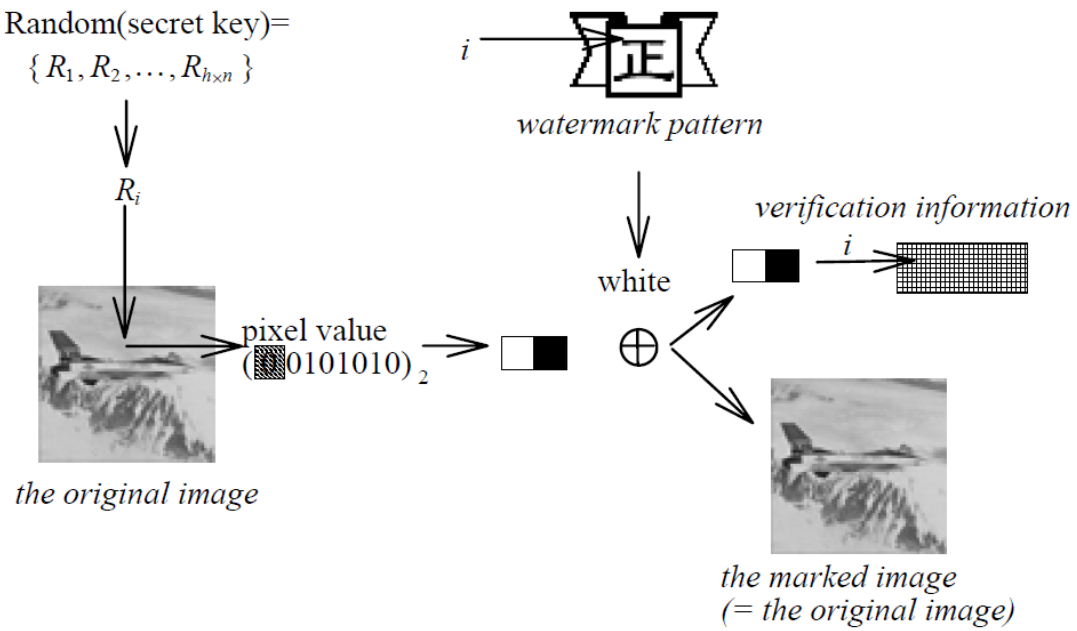


FIG. 3.1 – Processo de marcação de imagem (extraído de (HWANG, 2000))

Os processos de cifragem e decifragem usados por (HWANG, 2000) são descritos nos passos abaixo, considerando que a imagem *M* possui dimensão *k x l* e a marca d'agua *P* possui dimensão *h x n*:

Cifragem:

1º Passo: Selecionar um número aleatório *S* como chave secreta da imagem *M*;

2º Passo: Usar *S* como semente para *h x n* diferentes números aleatórios no intervalo [0, *k x l*]. R_i denota o *i*-ésimo número aleatório;

3º Passo: Atribuir o *i*-ésimo par (v_{i1}, v_{i2}) da informação de verificação *V* baseado na tabela TAB. 3.1.

4º Passo: Juntar todos os pares (v_{i1}, v_{i2}) para construir a informação de verificação *V*.

Decifragem:

1º Passo: Usar *S* como semente para gerar *h x n* diferentes números aleatórios (R_i);

2º Passo: Atribuir a cor do *i*-ésimo pixel do padrão de Marca d'água *P'* baseado na imagem *F* da seguinte forma:

i. Pegar o bit mais a esquerda, *b*, do pixel R_i da imagem *F*. Se *b* for "1", então defina f_i = (1,0); senão f_i = (0,1).
ii. Se f_i for igual ao i-ésimo par de *V*, então atribua branco como a cor do *i*-ésimo pixel de *P'*, senão, atribua preto.

3º Passo: Se *P'* puder ser reconhecido como *P* pelo sistema visual humano, a organização deve julgar que a imagem *F* é uma cópia de *M*.

TAB. 3.1 - Regras para atribuição da informação de verificação

Cor do *i*-ésimo pixel de *P*	O bit mais a esquerda do pixel R_i de M	Par (v_{i1}, v_{i2}) de *V*
Preto	1	(0,1)
Preto	0	(1,0)
Branco	1	(1,0)
Branco	0	(0,1)

4 CRIPTOGRAFIA VISUAL E ESTEGANOGRAFIA

Criptografia visual e esteganografia podem ser aplicadas sucessivamente a um segredo de forma a aumentar a segurança, conforme proposto por (JITHESH & KUMAR, 2005). Nesse caso, o segredo é inicialmente cifrado com criptografia visual, gerando *n* transparências. Em seguida, essas transparências seriam ocultadas por esteganografia, podendo todas as transparências serem ocultadas sob a mesma imagem ou cada transparência sob uma imagem diferente. Em (GOKUL et al, 2012) é descrita uma técnica multicamadas de marca d'água para incorporar uma mensagem secreta em uma imagem usando criptografia visual e seleção do bit menos significativo, SLSB *(Selected Least Significant Bit).* A metodologia proposta é:

Cifragem

1º passo: O texto secreto é escrito em uma imagem. Essa imagem é a entrada do algoritmo de geração de transparências.

2º passo: O algorimo retorna duas transparências.

3º passo: As transparências, juntamente com a imagem de capa, são dadas como entrada do método de cifragem SLSB.

4º passo: O resultado do método SLSB é a imagem de capa com mudanças mínimas

Decifragem

1º passo: A imagem esteganografada é dada como entrada para o SLSB.

2º passo: Duas transparências são obtidas como saída do SLSB

3º passo: As transparências formam a entrada do algoritmo de combinação de transparências

4º passo: O resultado do passo 3 é uma imagem com a mensagem secreta

Esse tipo de aplicação multicamada é diferente do proposto pelo presente trabalho. Ao invés de aplicar a criptografia visual e posteriormente esconder os *shares* por um processo de esteganografia, neste trabalho a criptografia visual e a esteganografia serão aplicadas concomitantemente, resultando em um método híbrido, como será descrito no próximo capítulo.

5 SISTEMA PROPOSTO

5.1 APLICAÇÃO ESCOLHIDA

Sempre foi uma grande preocupação de governos a segurança das informações envolvidas com a governabilidade. Uma das técnicas mais antigas para evitar que mensagens sejam lidas por terceiros é a esteganografia, que consiste em esconder a existência da mensagem. Nesse contexto, este trabalho desenvolverá um sistema de esteganografia com criptografia visual com potencial capacidade de ser usado pelo Exército Brasileiro como forma de aumentar a segurança na troca de mensagens.

De forma semelhante à marca d'água, um texto será "escondido" em uma imagem de cobertura sem alterar seus pixels, gerando uma imagem de verificação. Dessa forma, somente com a imagem marcada, não será possível identificar que uma mensagem foi escondida ali e o receptor só poderá ler a mensagem de posse também da imagem de verificação e da chave secreta. Isso não só torna a mensagem secreta como, de certa forma, autenticada.

Na cifragem, como ilustra a figura FIG. 5.1, a partir da mensagem em forma de imagem *M,* da chave secreta *k* e da imagem de cobertura *C*, que será usada para esconder *M*, obtém-se a imagem de verificação *V*.

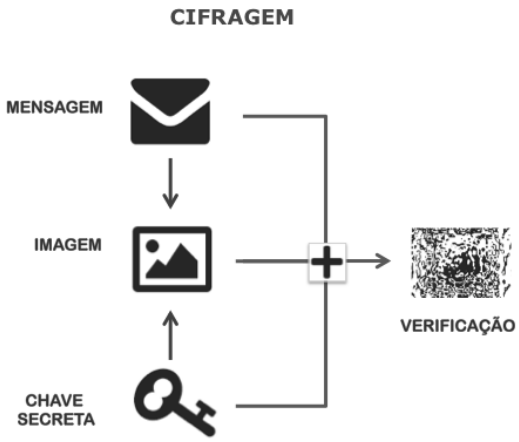


FIG. 5.1 – Cifragem no sistema proposto

Na decifragem, como ilustra a figura FIG. 5.2, de posse das duas imagens C e V , e da chave secreta k, é possível recuperar a mensagem M.

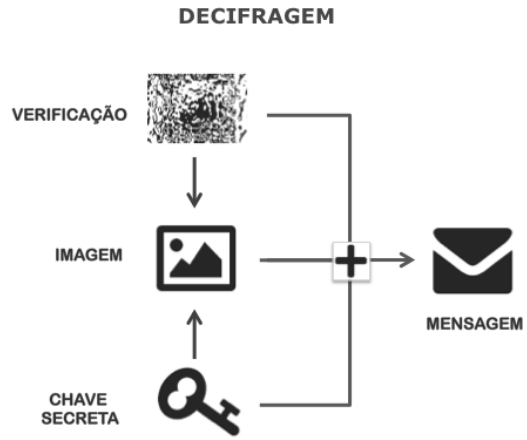


FIG. 5.2 – Decifragem no sistema proposto

5.2 ESPECIFICAÇÃO DO ALGORITMO UTILIZADO

Nesta seção, juntamente com a especificação do sistema proposto, será ilustrada sua aplicação a um pequeno exemplo.

5.2.1 CIFRAGEM

A cifragem no sistema proposto se divide em duas partes. A primeira consiste na criação da imagem de verificação, *V*, a partir da mensagem na forma de imagem, *M*, da imagem de cobertura usada para escondê-la, *C*, e de uma chave inteira. Na segunda etapa, a imagem de verificação *V* gerada passará por transformações invertíveis com o objetivo de impedir que informações possam ser obtidas visualmente a partir da mesma.

Para ilustrar o procedimento usaremos como mensagem a figura FIG. 5.3 e como imagem original, a figura FIG. 5.4. Em cada uma delas os quadrados pretos e brancos representam pixels. A chave escolhida foi o número “5”.

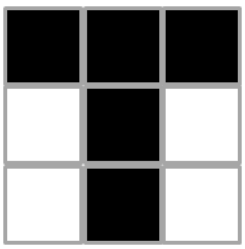


FIG. 5.3 – Mensagem

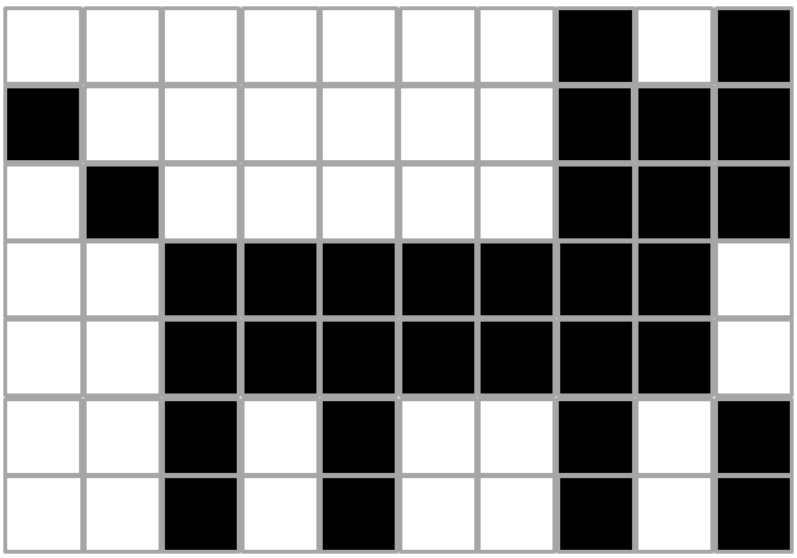


FIG. 5.4 – Imagem de Cobertura

5.2.1.1 CRIAÇÃO DA IMAGEM DE VERIFICAÇÃO

A criação da imagem de verificação, *V*, é feita de forma similar ao descrito no Capítulo 3 para sistemas de marca d'água:

1º Passo: Selecionar um inteiro aleatório *k* como chave secreta, que posteriormente será enviado ao receptor ou será combinada previamente.

No caso do exemplo foi escolhido o número "5";

2º Passo: Usar *k* como semente para $W_m \times H_m$ (dimensão de *M*) gerar números pseudo-aleatórios no intervalo [0, $W_i \times H_i$], em que $W_i \times H_i$ é a dimensão de *I*. R_i denota o *i*-ésimo número aleatório.

No exemplo $W_m \times H_m = 3x3 = 9$ e $W_i \times H_i = 10x7 = 70$. A sequência gerada pela classe geradora de pseudo-aleatórios do C#, *Random*, foi: { 23, 19, 18, 43, 32, 64, 10, 66, 41};

3º Passo: Atribuir o *i*-ésimo par (v_{i1}, v_{i2}) da transparência baseado na tabela TAB. 5.1.

A figura FIG. 5.5 mostra a imagem base numerada e com os pixels que serão usados marcados em vermelho.

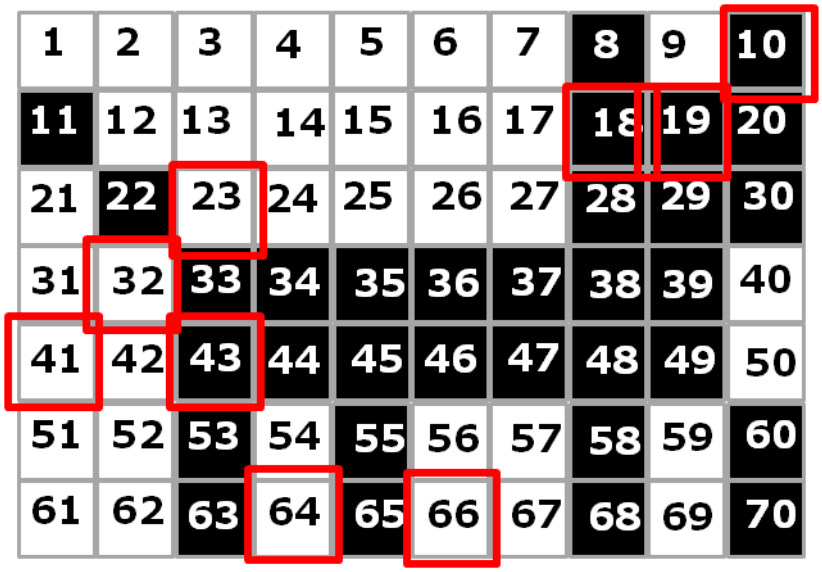


FIG. 5.5 –Seleção dos pixels da imagem de cobertura a serem utilizados

TAB. 5.1 - Regras para criação da transparência

Cor do *i*-ésimo pixel de *M*	O bit mais a esquerda de pixel R_i de *I*	Par (v_{i1}, v_{i2}) de *V*
Preto	1	(0,1)
Preto	0	(1,0)
Branco	1	(1,0)
Branco	0	(0,1)

4º Passo: Juntar todos os pares (v_{i1}, v_{i2}) para construir a imagem de verificação *V*.

A imagem gerada pelo exemplo é mostrada na figura FIG. 5.6. Observe que o primeiro pixel da mensagem da figura FIG. 5.3 é preto e o pixel 23 (R_1) é branco. No sistema RGB, o primeiro bit de pixels brancos é 1 e de pixels pretos é zero. Portanto, esse caso corresponde a primeira linha da tabela TAB. 5.1 e por isso o primeiro par de V é (0,1) = (preto, branco).

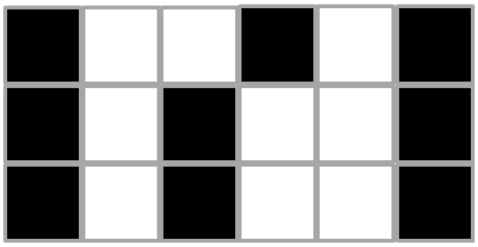


FIG. 5.6 –Imagem de verificação gerada

5.2.1.2 TRANSFORMAÇÃO DA IMAGEM DE VERIFICAÇÃO

Obtida a transparência, serão aplicadas três transformações invertíveis sobre a mesma:

1º passo: Transpor a matriz de pixels de *V*,gerando V_1;

2º passo: Fazer permutações aleatórias de colunas da matriz de pixel de V_1, gerando V_2 . O número de permutações é igual ao valor absoluto da largura de V_1 e as colunas permutadas serão escolhidas a partir de uma sequência pseudo-aleatória gerada pela mesma semente *k* usada no processo de criação de *V* e pelas dimensões da imagem base w_i e h_i; A semente será $(k\ W_i)mod\ H_i$

3º passo: Transpor V_2, gerando V_3, que será enviada juntamente com a chave k e a imagem I para o destinatário da mensagem escondida.

5.2.2 DECIFRAGEM

A decifragem do sistema prosposto é o processo inverso da cifragem. Primeiramente, deve-se aplicar as transformações inversas das aplicadas na segunda parte da cifragem, recuperando a imagem de verificação original V. Em seguida, deve ser aplicado um processo de decifragem semelhante ao descrito no Capítulo 3.

5.2.2.1 RECUPERAÇÃO DA IMAGEM DE VERIFICAÇÃO *V*

De posse da imagem de verificação V_3 e da chave secreta k, pode-se recuperar V com os seguintes passos:

1º passo: Transpor V_3 , obtendo-se V_2;

2º passo: Gerar uma sequência de $2*W_v$ números pseudo-aleatórios no intervalo $[0, W_v]$ com a semente $(k\ W_i)mod\ H_i$, em que W_v é a largura de V_2. A sequência inversa da obtida determina as colunas a serem permutadas para se recuperar V_1;

3º passo: Transpor V_1 , obtendo-se V.

5.2.2.2 RECUPERAÇÃO DA MENSAGEM *M*

A partir da informação de verificação V, da chave secreta k e da imagem I, é possível recuperar M com os seguintes passos:

1º Passo: Usar *k* como semente para gerar $W_v \times H_v$ diferentes números aleatórios (R_i). A sequência gerada é a mensa da cifragem: { 23, 19, 18, 43, 32, 64, 10, 66, 41};

2º Passo: Atribuir a cor do *i*-ésimo pixel do padrão da mensagem *M* baseado na imagem *I* da seguinte forma:

i. Pegar o bit mais à esquerda, *b*, do pixel R_i da imagem *I*. Se *b* for "1", então defina $f_i = (1,0)$; senão $f_i = (0,1)$.
   No caso do exemplo, o pixel 23 é branco, portanto $f_1 = (1,0)$.

ii. Se f_i for igual ao i-ésimo par de *V*, então atribua branco como a cor do *i*-ésimo pixel de *M*, senão, atribua preto.
   No exemplo, o primeiro par da verificação é (0,1) , que é diferente de f_1. Logo, o primeiro pixel da mensagem será preto.

A mensagem *M* gerada, pode divergir ligeiramente (alguns pixels) da mensagem escrita, mas o conteúdo da mensagem será reconhecido pelo sistema visual humano. No exemplo mostrado a mensagem recuperada é exatamente igual à mensagem inicial mostrada na figura FIG. 5.3.

5.3 FERRAMENTAS UTILIZADAS

5.3.1 LINGUAGEM DE PROGRAMAÇÃO

O sistema proposto será implementado em C#, utilizando as classes de tratamento de imagem *Image* e *Bitmap*, disponíveis em (Image Class, 2013).

Inicialmente considerou-se o uso da linguagem Ruby, devido a sua facilidade e clareza de código. Entretanto, a *gem* de processamento de imagem necessária, RMagick (RMagick, 2013), não pode ser usada. Sua instalação em ambos os sistemas operacionais Windows 7 e Mac OSX 10.8 se mostrou problemática, pois existe mais de um documento descrevento o procedimento de instalação e não foi obtido sucesso com nenhum deles.

O ambiente C#, entretando, já foi configurado automaticamente, com a instalação da IDE gratuita *Visual Studio Express 2010* (Visual Studio Express, 2013). Além disso, ambas as autoras desse trabalho já possuiam experiência com programação em C#, ao contrário de Ruby.

5.3.2 REPOSITÓRIO

O código será armazenado em um repositório público e gratuito do GitHub (GitHub, 2013), chamado de Camaleão (Camaleão, 2013). O nome escolhido é uma alusão a capacidade de se camuflar do camaleão, que remete a ideia da mensagem escondida por esteganografia.

A escolha do repositório público também foi causada pelo desejo de transformar o presente trabalho em um projeto *open source* que possa ser melhorado e referenciado por trabalhos futuros.

5.3.3 CONTROLE DE TAREFAS

Para a definição e controle de tarefas realizadas, foi escolhido o Trello (Trello, 2013), uma ferramenta online e gratuita de colaboração que permite a criação de quadros para a organização de micro-tarefas, além de outras funcionalidades.

6 IMPLEMENTAÇÃO DO SISTEMA PROPOSTO

6.1 IMPLEMENTAÇÃO INICIAL

Para estabelecer um contato inicial com a manipulação de imagens pelas classes em *C#* de tratamento de imagens *Image* e *Bitmap* (Image Class, 2013), inicialmente foi feita uma implementação do método *2 out of 2* de Naor e Shamir. O algoritmo desenvolvido é baseado na seção 2.1.1 utilizando-se o mapeamento em 4 sub-pixels, mostrado na tabela TAB. 2.2.

Foram criadas duas classes, uma para cifragem, a *NaorShamirCypher*, e uma para a decifragem, a *NaorShamirDecypher*. A primeira contém um método que gera as duas transparências a partir de uma imagem, e a segunda contém um método que realiza a sobreposição das duas transparências e gera a mensagem recuperada. O código em *C#* de ambas as classes estão disponíveis no Apêndice 1. Abaixo estão imagens usadas para testes das classes implementadas. A figura FIG. 6.1 abaixo é a imagem a ser compartilhada, e a figura FIG. 6.2 mostra o resultado da sobreposição das duas transparências geradas, que são as figuras FIG. 6.3 e FIG. 6.4.

Através da criação desses métodos pode-se notar que a manipulação de imagens tratando diretamente da cor e posicionamento de seus pixels é muito facilitada pelos métodos da classe *Bitmap.*

FIG. 6.1 - Imagem a ser compartilhada.

FIG. 6.2 - Imagem recuperada sobrepondo-se as duas transparências.

FIG. 6.3 - Transparência número 1.

FIG. 6.4 - Transparência número 2.

6.2 IMPLEMENTAÇÃO DO SISTEMA PROPOSTO

Foi feita a implementação do algoritmo descrito na seção 5.2 na linguagem *C#*, também utilizando as classes *Image* e *Bitmap* (Image Class, 2013). Foram criadas três classes, uma para cifragem, a *Cypher*, uma para a decifragem, a *Decypher*, e uma para métodos comuns às duas classes, a *Utils*.

A classe *Cypher* implementa a cifragem conforme é descrito na seção 5.2.1. Ela possui o método *GenerateShare()* que cria a imagem de verificação. Este método utiliza um outro método também desta classe chamado *TransformShare()* que realiza a transformação da imagem de verificação, conforme a seção 5.2.1.2.

A classe *Decypher* implementa a decifragem conforme é descrito na seção 5.2.2. Ela possui o método *GetMessage()* que recupera a mensagem a partir da imagem de verificação. Este método utiliza um outro método também desta classe chamado *TransformShareBack()* que realiza recuperação da imagem de verificação (transformação inversa), conforme a seção 5.2.2.1.

A classe *Utils* possui dois métodos estáticos que são utilizados pelas duas classes já descritas. O método *TransposeImage(Bitmap image)* realiza a transposição da imagem *image* e é utilizado pelos métodos *TransformShare()* e *TransformShareBack()*. O método *CreatePrivateKey(Random random, int maxValue, int keyLength)* coloca em uma lista de inteiros os números pseudoaleatórios gerados pelo randômico *random*, com máximo valor de cada número igual a *maxValue* e o tamanho da lista igual a *keyLength*. Este método é utilizado pelos métodos *GenerateShare()* e *GetMessage()*, com o randômico que tem por semente a chave secreta.

O código em *C#* de ambas as classes está disponível no Apêndice 2. Abaixo estão imagens usadas para testes das classes implementadas. As figuras FIG. 6.5, FIG. 6.6 e FIG. 6.7 correspondem a imagem base, a mensagem e a imagem de verificação, respectivamente. Foi utilizada a chave secreta $k = 7$. A figura FIG. 6.8 mostra a mensagem recuperada pelo receptor. Foram adicionadas bordas na mensagem original e na mensagem recuperada para

deixar nítidos os limites das imagens. Observação: o fundo da figura FIG. 6.5 parece cinza mas tratam-se de pixels pretos e brancos igualmente distribuídos.

FIG. 6.5 - Imagem base (dimensão original: 2048 × 1934).

Em 1994, Adi Shamir e Moni Naor apresentaram um novo modelo de criptografia seguro e fácil de usar, a criptografia visual. Neste trabalho é implementado um sistema criptográfico de esteganografia combinado com criptografia visual baseado em um sistema de marca d'água Diferentemente dos outros trabalhos pesquisados, que aplicam criptografia visual e posteriormente a esteganografia, esse trabalho propõe o uso das duas técnicas simultaneamente, de tal modo que a imagem que "esconderia" a mensagem pode ser interpretada como uma da transparências do modelo de Naor e Shamir.

FIG. 6.6 - Mensagem (dimensão original: 743 × 259).

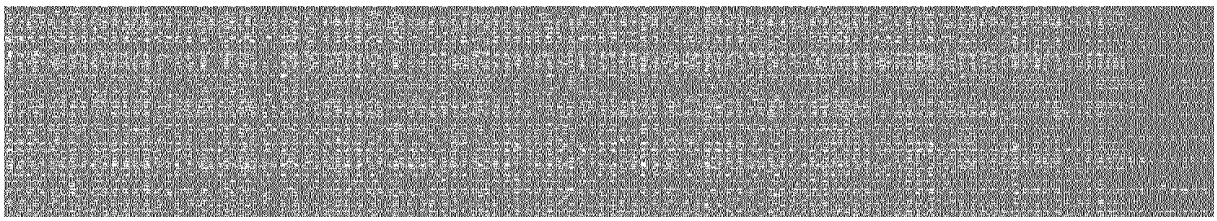


FIG. 6.7 - Imagem de verificação (dimensão original: 1486 × 259).

Em 1994, Adi Shamir e Moni Naor apresentaram um novo modelo de criptografia seguro e fácil de usar, a criptografia visual. Neste trabalho é implementado um sistema criptográfico de esteganografia combinado com criptografia visual baseado em um sistema de marca d'água Diferentemente dos outros trabalhos pesquisados, que aplicam criptografia visual e posteriormente a esteganografia, esse trabalho propõe o uso das duas técnicas simultaneamente, de tal modo que a imagem que "esconderia" a mensagem pode ser interpretada como uma da transparências do modelo de Naor e Shamir.

FIG. 6.8 - Mensagem recuperada (dimensão original: 743 × 259).

7 ESCOLHA DAS IMAGENS

Para que o sistema seja eficaz, a mensagem recuperada deve ser legível e deve conter o mínimo de ruído possível. Neste capítulo é feita uma análise de como seriam a melhor imagem base e a melhor imagem para representar a mensagem, de forma a atender tais requisitos. É importante ressaltar que as imagens mostradas neste capítulo não estão em seu tamanho original e foram redimensionadas para melhor visualização dos resultados das análises realizadas. Além disso, foram adicionadas bordas nas mensagens e nas mensagens recuperadas para deixar nítidos os limites das imagens.

7.1 MENSAGEM

Primeiramente analisou-se o tipo de fonte a ser utilizada na mensagem. Notou-se que fontes menos espessas (mais finas) e não em negrito proporcionam uma mensagem recuperada mais legível, como pode-se notar nas figuras de FIG. 7.1 a FIG. 7.4. Foi escolhida então a fonte Courier New.

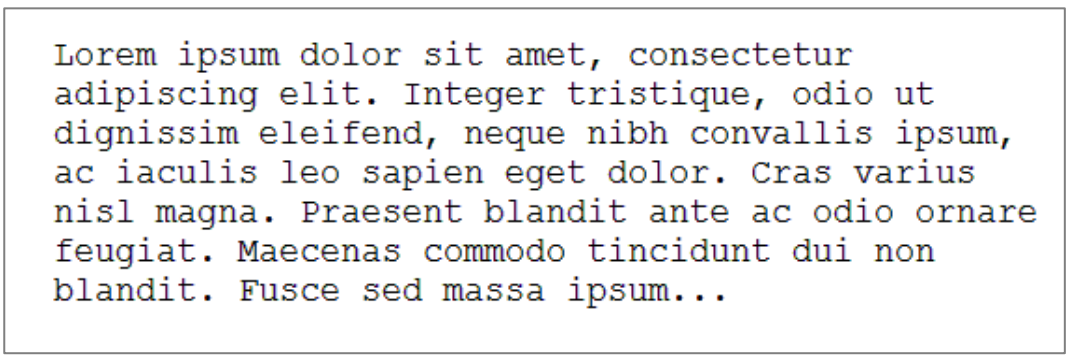


FIG. 7.1 - Mensagem com fonte fina.

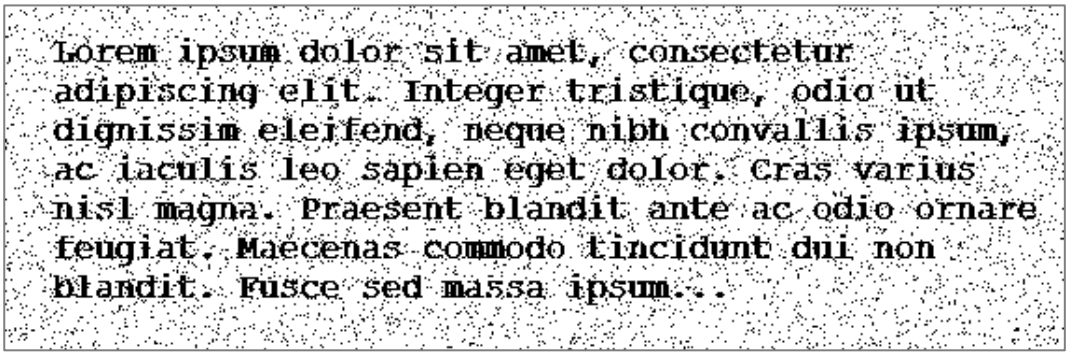


FIG. 7.2 - Mensagem recuperada da mensagem da FIG. 7.1.

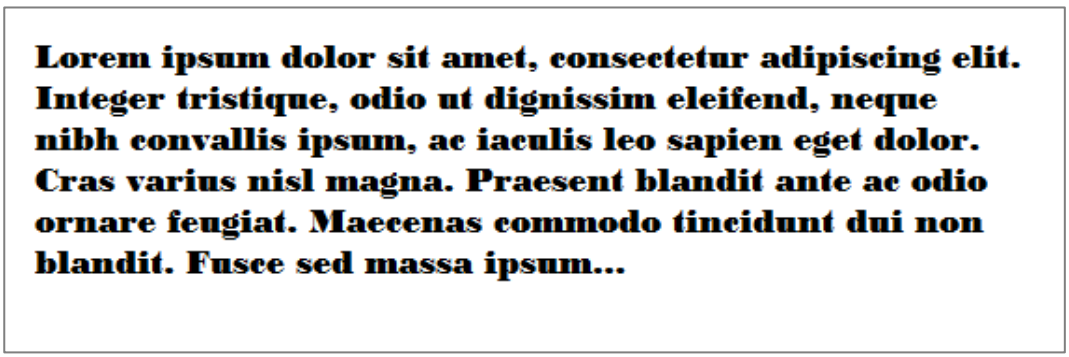


FIG. 7.3 - Mensagem com fonte grossa.

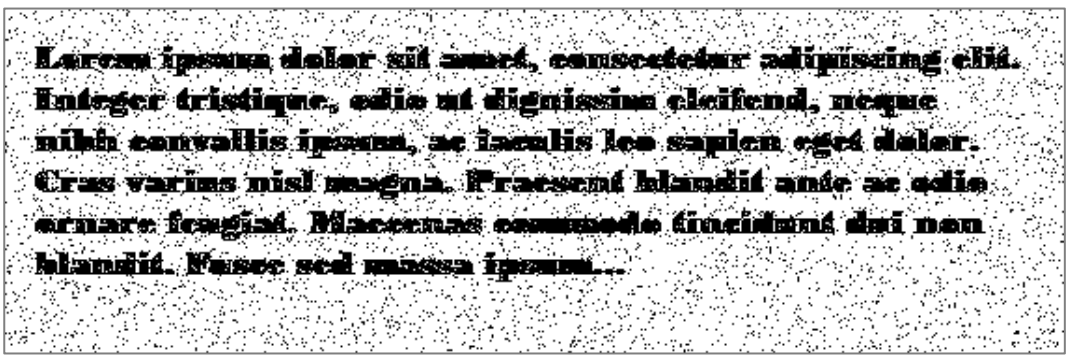


FIG. 7.4 - Mensagem recuperada da mensagem da FIG. 7.3.

A partir da fonte escolhida, analisou-se então o tamanho da letra. Nota-se que quanto maior a fonte, mais nítida a mensagem recuperada se torna, como pode ser observado nas imagens de FIG. 7.5 a FIG. 7.8. Entretanto, notou-se que não é necessária uma fonte muito grande para atingir uma mensagem legível. Para não gastar espaço desnecessariamente com fontes muito grandes, escolheu-se então o tamanho de fonte 16.

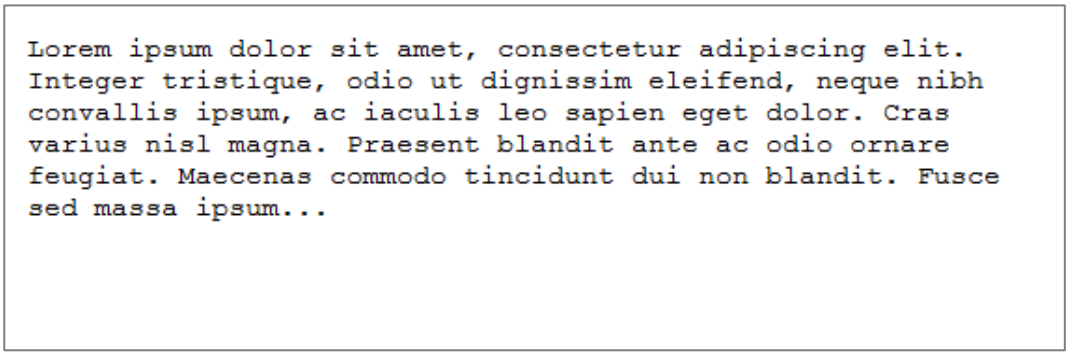


FIG. 7.5 - Mensagem com fonte pequena (Courier New 11).

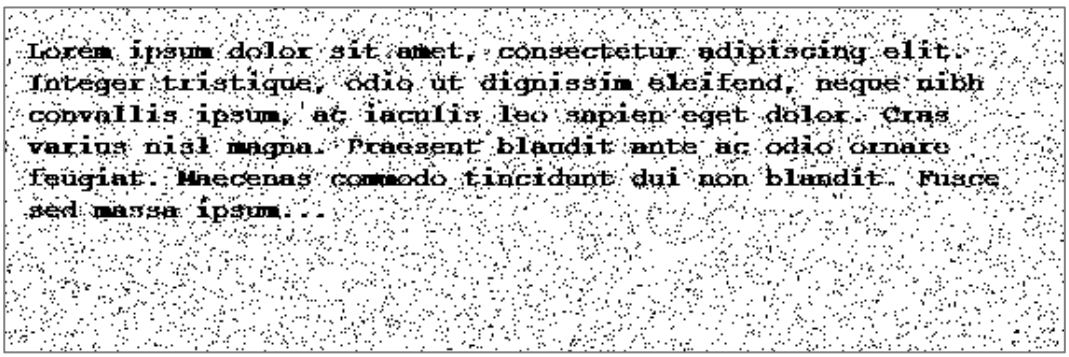


FIG. 7.6 - Mensagem recuperada da mensagem da FIG. 7.5.

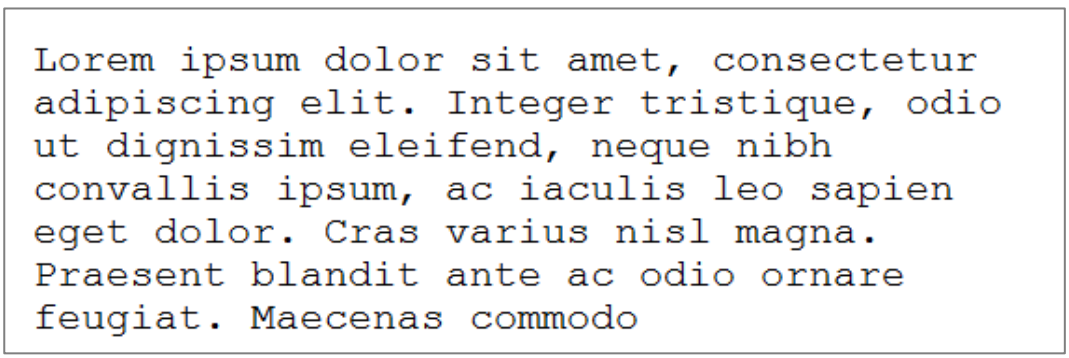


FIG. 7.7 - Mensagem com fonte grande (Courier New 16).

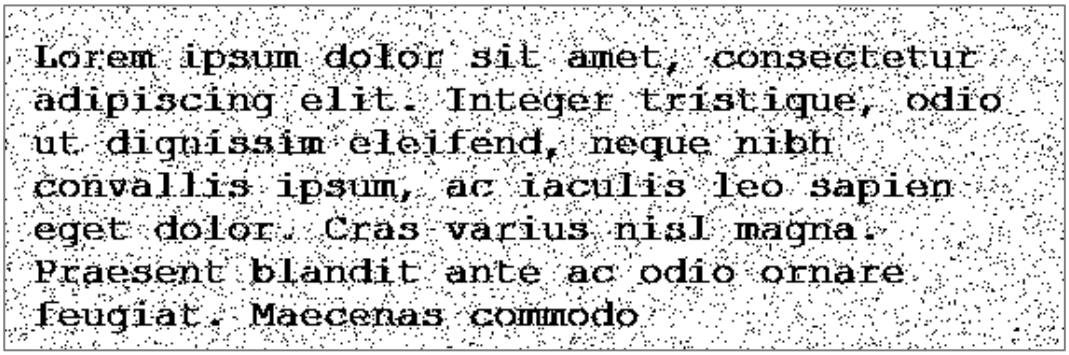


FIG. 7.8 - Mensagem recuperada da mensagem da FIG. 7.7.

O tamanho do texto a ser transmitido não afeta a mensagem recuperada. Entretanto, o ideal é que se divida um texto grande em várias partes e se transmitam essas partes em mensagem separadas para dificultar a recuperação do texto completo em uma interceptação.

O tamanho da mensagem (imagem onde estará o texto) depende apenas do tamanho do próprio texto, pois notou-se que o ideal é que o texto da mensagem ocupe a maior área possível da imagem em que será inserida. Isso deixa a imagem de verificação mais ruidosa, não deixando visível que há alguma informação escondida ali. Pode-se observar isso nas figuras de FIG. 7.9 a FIG. 7.12 abaixo: na primeira verificação percebe-se que que há uma "borda" sem informação e toda a informação se concentra em uma área da imagem, e na segunda isso fica menos visível e a verificação fica mais ruidosa.

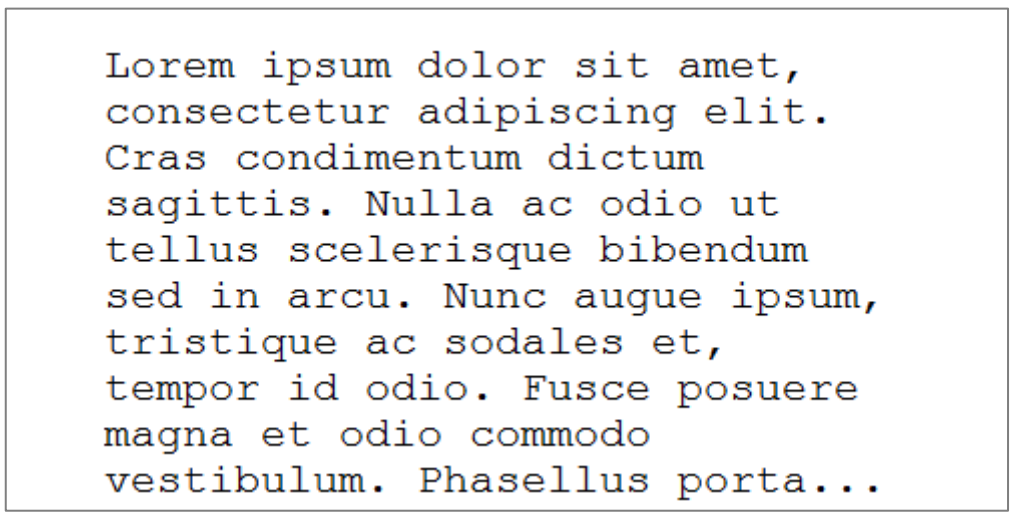


FIG. 7.9 – Mensagem com texto ocupando uma área central da imagem.

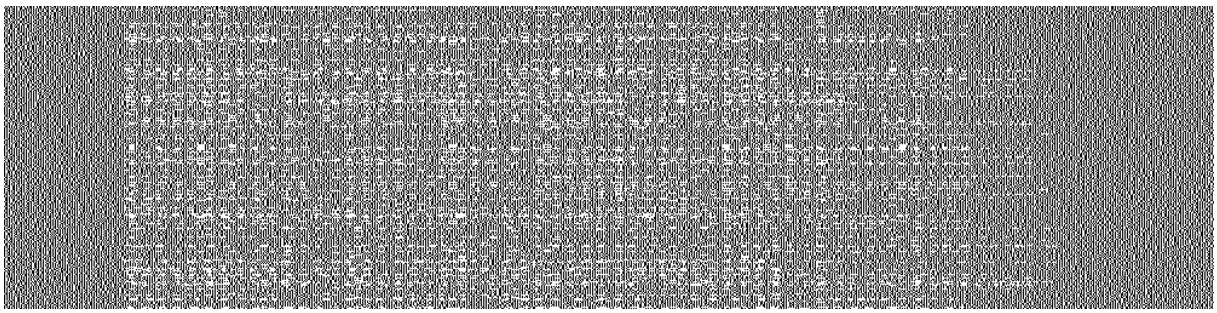


FIG. 7.10 - Imagem de verificação com da mensagem da FIG. 7.9.

Lorem ipsum dolor sit amet,
consectetur adipiscing elit. Cras
condimentum dictum sagittis. Nulla ac
odio ut tellus scelerisque bibendum
sed in arcu. Nunc augue ipsum,
tristique ac sodales et, tempor id
odio. Fusce posuere magna et odio
commodo vestibulum. Phasellus porta
tempus nunc sit amet aliquet. Etiam
nec felis felis. Maecenas urna leo,
hendrerit eget egestas a, consequat...

FIG. 7.11 – Mensagem com texto ocupando a maior área possível da imagem.

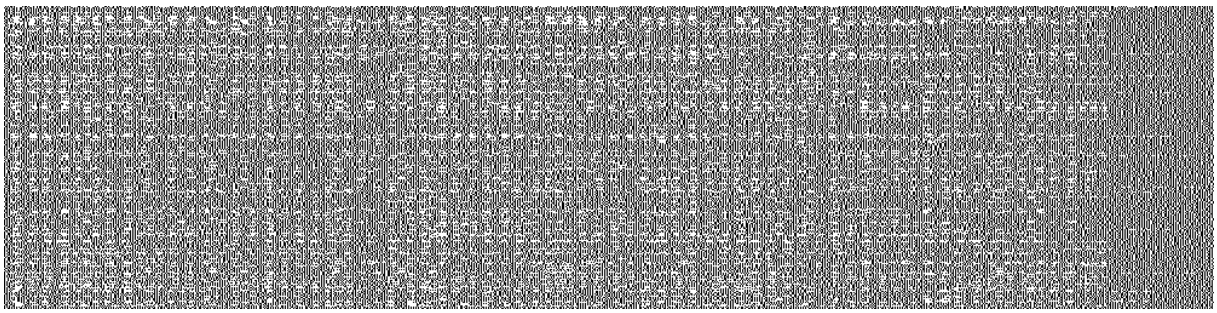


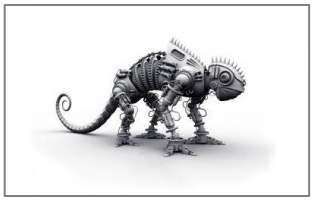
FIG. 7.12 - Imagem de verificação da mensagem da FIG. 7.11.

7.2 IMAGEM BASE

Para as análises da imagem base utilizou-se a mensagem com fonte Courier New tamanho 16. Primeiramente analisou-se a proporção entre pixels brancos e pretos na imagem.

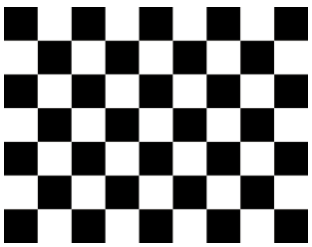
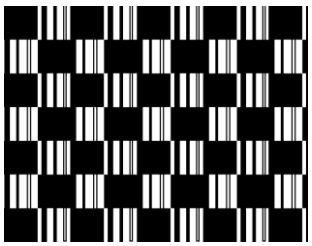
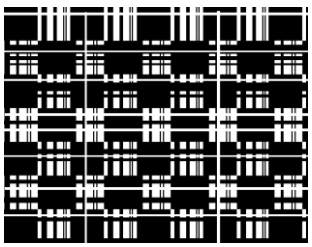
Notou-se que imagens com pixels predominantemente pretos geram uma mensagem recuperada mais nítida, e quanto maior a proporção de pixels brancos mais ruído aparecia na mensagem recuperada. Isso pode ser visto na tabela TAB. 7.1 abaixo, onde nota-se que a melhor mensagem recuperada foi gerada pela imagem base Camaleao3. Observação: o fundo da imagem Camaleao3 parece cinza pois foi redimensionada, mas tratam-se de pixels pretos e brancos igualmente distribuídos.

TAB. 7.1 - Imagens com diferentes proporções entre pixels brancos e pretos, e suas respectivas mensagens recuperadas.

Nome	Imagem	Mensagem Recuperada
Camaleao1		Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer tristique, odio ut dignissim eleifend, neque nibh convallis ipsum, ac iaculis leo sapien eget dolor. Cras varius nisl magna. Praesent blandit ante ac odio ornare feugiat. Maecenas commodo
Camaleao2		Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer tristique, odio ut dignissim eleifend, neque nibh convallis ipsum, ac iaculis leo sapien eget dolor. Cras varius nisl magna. Praesent blandit ante ac odio ornare feugiat. Maecenas commodo
Camaleao3		Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer tristique, odio ut dignissim eleifend, neque nibh convallis ipsum, ac iaculis leo sapien eget dolor. Cras varius nisl magna. Praesent blandit ante ac odio ornare feugiat. Maecenas commodo

Analisou-se também quantidade de detalhes na imagem. Pela tabela TAB. 7.2 abaixo, pode-se perceber que uma imagem com poucos detalhes, como a Xadrez1, proporciona uma mensagem recuperada com menos ruído.

TAB. 7.2 - Imagens com quantidades de detalhes diferentes, e suas respectivas mensagens recuperadas.

Nome	Imagem	Mensagem Recuperada
Xadez1		Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer tristique, odio ut dignissim eleifend, neque nibh convallis ipsum, ac iaculis leo sapien eget dolor. Cras varius nisl magna. Praesent blandit ante ac odio ornare feugiat. Maecenas commodo
Xadez2		Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer tristique, odio ut dignissim eleifend, neque nibh convallis ipsum, ac iaculis leo sapien eget dolor. Cras varius nisl magna. Praesent blandit ante ac odio ornare feugiat. Maecenas commodo
Xadrez3		Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer tristique, odio ut dignissim eleifend, neque nibh convallis ipsum, ac iaculis leo sapien eget dolor. Cras varius nisl magna. Praesent blandit ante ac odio ornare feugiat. Maecenas commodo
Xadrez4		Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer tristique, odio ut dignissim eleifend, neque nibh convallis ipsum, ac iaculis leo sapien eget dolor. Cras varius nisl magna. Praesent blandit ante ac odio ornare feugiat. Maecenas commodo

Portanto, a imagem base ideal poderia ser a Camaleao3 ou a Xadrez1. Entretanto, Xadrez1 é muito simples e poderia contribuir para a quebra do sistema, então optou-se pela Camaleao3.

8 ANÁLISE DE SEGURANÇA

8.1 SEQUÊNCIA PSEUDO-ALEATÓRIA

No presente trabalho é usada a classe *Random* da linguagem *C#* para a geração das sequências pseudo-aleatórias a partir de uma semente. Nessa classe, duas sequências geradas com a mesma semente e com o mesmo intervalo de valores são iguais. A implementação é baseada no algoritmo subtrativo de Donald E. Knuth's. (Random Class, 2013).

Um gerador subtrativo calcula uma sequência de números pseudo-aleatórios onde cada número é congruente a subtração dois números anteriores da sequência conforme a recursão abaixo, para *i* , *j*, e *m* inteiros positivos:

$$r_n = (r_{n-i} - r_{n-j}) mod\ m$$

Essa fórmula relaciona r_n , r_{n-i} e r_{n-j} e esses números não são realmente independentes como números aleatórios deveriam ser, portanto a partir da observação de *i* números consecutivos pode-se predizer números seguintes. (Subtractive Generator, 2013)

O Laboratório de Tecnologia da Informação (ITL) do Instituto Nacional de Padrões e Tecnologia (NIST) fornece liderança técnica para os Estados Unidos provendo padrões de medição e infraestrutura. O artigo (RUKHIN et al, 2010), publicado pelo NIST, discute aspectos importantes para a seleção e teste de geradores de sequências aleatórias e pseudo-aleatórias para aplicações criptográficas.

Para verificar a aleatoriedade do gerador da classe *Random*, foi utilizado o Teste de Frequência Monobit do NIST, implementado em (TORRES, 2010). O teste tem como entrada uma string de zeros e uns de comprimento maior que 100. O arquivo RandomC#.txt, usado

como entrada do programa, contém a seguinte sequência de bits, gerada pela classe *Random* do *C#*:

00001011000011110000001001110101100110110000010110100011001100011010111110000111110111011100011101001010001011111000001011110000010001011110010101101011001100010101101101101011101000110001100111000011100110010001010101110111001101010110011111011000110010011001111011011101110100000111111010001010101100000011010001011101111100001000011101111010110110111011011010100100000011111001101000011100110000100001000011100011001111101010010111000101111111111001110110101000111110010111100100100111101110111010000111101100001011000110110110000000001010000011001101110001100101011101010111000100000001110111101001111
1

O resultado do teste é mostrado na figura FIG. 8.1 abaixo.

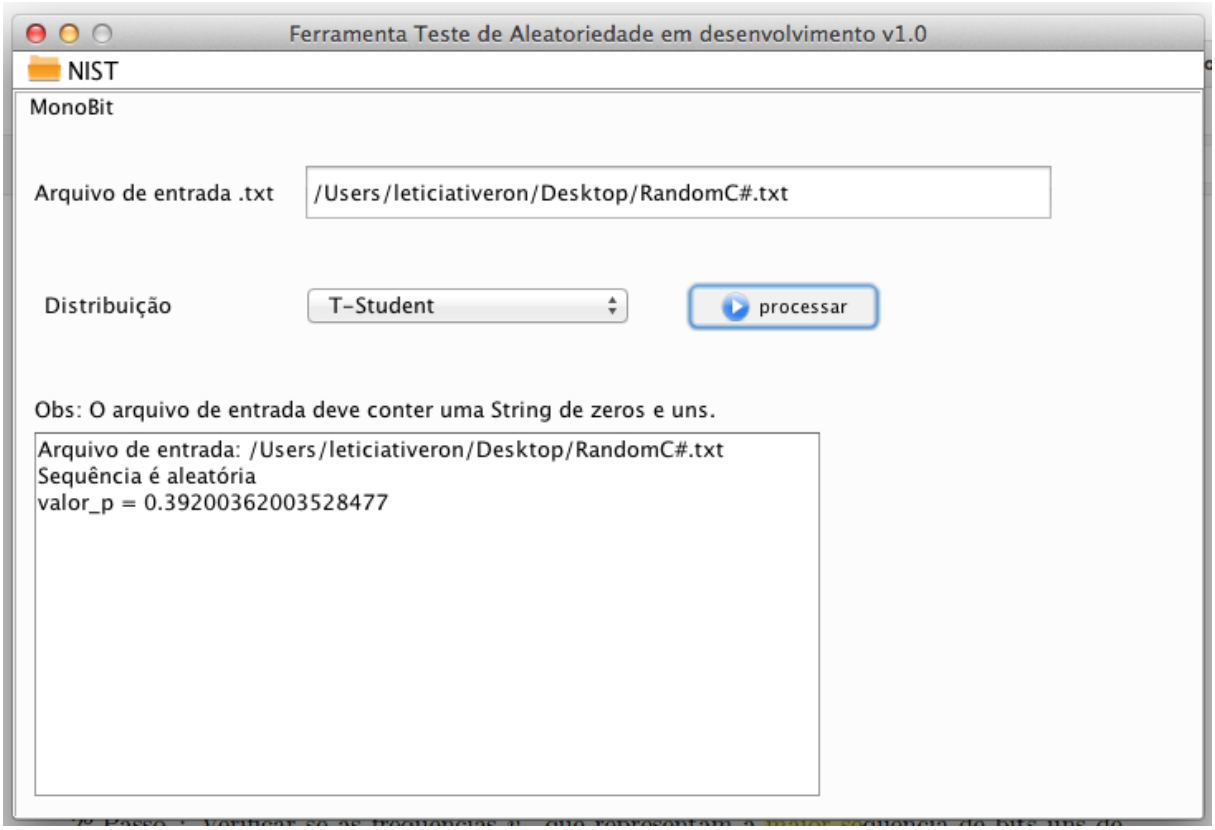


FIG. 8.1 - Tela da ferramenta de teste de aleatoriedade implementado em (TORRES, 2010) mostrando o resultado do teste realizado.

O objetivo do teste monobit é verificar em uma sequência de bits de entrada, se a frequência de zeros e uns é a mesma, ou seja, se a proporção é de 0,5. Em caso positivo, a sequência pode ser considerada aleatória. A análise é feita através do *valor_p* encontrado, se *valor_p* > 0,01, a sequência pode ser considerada aleatória. O valor encontrado para a sequência gerada pela classe *Random* foi 0,392 > 0,01. Portanto, a sequência pode ser considerada aleatória.

Esse é o primeiro teste de quinze propostos em (RUKHIN et al, 2010) e é considerado um teste eliminatório. Para uma análise mais profunda da aleatóriedade geralmente são realizados todos os testes, porém, como a sequência construída não é crítica para a segurança do sistema, o resultado positivo no primeiro teste foi considerado satisfatório. Além disso, A construção da sequência pseudo-aleatória pode ser facilmente alterada na implementação realizada e, portanto, pode ser substituída no futuro caso necessário.

8.2 QUEBRA DO SISTEMA

No sistema proposto o destinatário recebe três parâmetros: a chave secreta, a imagem base e a imagem de verificação. Para que se decifre a mensagem, é necessário que se disponha dos três elementos. Nesta seção será feita uma análise da segurança do sistema, verificando se seria possível, e o quão difícil seria, decifrar a mensagem com apenas dois dos três elementos.

8.2.1 POSSE DA CHAVE E DA IMAGEM DE VERIFICAÇÃO

Tendo a imagem de verificação, precisa-se recuperar a imagem de verificação original, sem a transformação. O tamanho da sequência pseudo-aleatória para o pré-processamento da imagem de verificação (*V*) é o dobro da largura da imagem de verificação e cada número da sequência varia de 0 até a largura de *V*. Logo, sendo *n* a largura de *V*, o número de sequências possíveis é n^{2n}.

Recuperada a imagem de verificação original, para decifrar a mensagem é necessário construir a sequência pseudo-aleatória para decifragem e a partir dela selecionar os pixels correspondentes da imagem base. Sem as dimensões da imagem base não é possível determinar essa sequência, porém podemos tentar força bruta nos $n/2 \times m$ pixels selecionados na imagem, em que n e m são respectivamente a largura e a altura da imagem de verificação. Cada pixel escolhido pode ser preto ou branco. Logo, existem $2^{nm/2}$ possibilidades para a escolha dos pixels.

Portanto, para o uso de força bruta, deverão ser realizados $n^{2n}2^{nm/2}$ testes.

8.2.2 POSSE DA IMAGEM BASE E DA CHAVE

De posse da chave secreta e da imagem é possível gerar uma sequência pseudo-aleatória com a mesma semente e limite da sequência necessária para decifragem. Porém não é possível determinar o tamanho da sequência uma vez que ele é obtido a partir das dimensões da imagem de verificação. Da mesma forma, sem a imagem de verificação também não é possível saber o tamanho da mensagem.

Nesse caso, para fazer o uso de força bruta é necessário que para cada valor possível de dimensão da mensagem ($n \times m$, com n e m naturais) sejam testados todos os valores possíveis para os pixels da imagem de verificação. Como cada pixel tem dois estados possíveis, preto ou branco, e o número de pixels da imagem de verificação é $2n \times m$, o número total de casos a ser testado é 2^{2mn}. Ainda, para cada imagem de verificação testada, é feito um processamento com tempo múltiplo da quantidade de pares de pixels da imagem de verificação, ou seja, multiplo de nm.

Portanto, a complexidade final para o uso de força bruta é $nm2^{2nm}nm = n^2m^2 2^{2mn}$.

8.2.3 POSSE DAS IMAGENS BASE E DE VERIFICAÇÃO

O tamanho da sequência pseudo-aleatória para o pré-processamento da imagem de verificação (*V*) é o dobro da largura da imagem de verificação e cada número da sequência varia de 0 até a largura de *V*. Logo, sendo n a largura de *V*, o número de sequências possíveis é n^{2n}.

Recuperada a imagem de verificação original, para decifrar a mensagem é necessário construir a sequência pseudo-aleatória para decifragem. Essa sequência tem $n/2 \times m$ números, em que n e m são respectivamente a largura e a altura da imagem de verificação. Cada número da sequência varia entre 0 e $w_i \times h_i$, em que w_i e h_i são respectivamente a largura e a altura da imagem base. Logo, existem $(w_i\ h_i)^{nm/2}$ possibilidades para essa sequência. Porém é menos custoso testar todas as sequências possíveis de seleção de pixels pretos ou brancos da imagem base, que é $2^{nm/2} < (w_i\ h_i)^{nm/2}$.

Portanto, para o uso de força bruta, deverão ser realizados $n^{2n}2^{nm/2}$ testes, assim como no caso 8.2.1.

8.2.4 ANÁLISE DA VIABILIDADE DA DECIFRAGEM

Na tabela TAB. 8.1 são mostrados os valores de complexidade calculados nas seções de 8.2.1 a 8.2.3, bem como o tempo total necessário para quebrar a cifra por força bruta. O tempo de 1ns para cada teste é bem menor que o tempo real medido nesse trabalho para a realização dos processos de cifragem e decifragem, que por vezes demoraram alguns segundos. Portanto, na prática, a força bruta para o sistema proposto é inviável.

TAB. 8.1 - Tempo de quebra do sistema por força bruta.

Caso	**Complexidade**	**Largura de V**	**Altura de V**	**Tempo de cada teste**	**Tempo Total**
8.2.1	$n^{2n}2^{nm/2}$	500	500	1ns	$>3*10^{9991}$ anos
8.2.2	$n^2m^2 2^{2mn}$	500	500	1ns	$>3*10^{9991}$ anos
8.2.3	$n^{2n}2^{nm/2}$	500	500	1ns	$>3*10^{9991}$ anos
Recuperação de *V*	n^{2n}	500	500	1 ns	$3*10^{2682}$ anos

9 CONCLUSÃO

9.1 CONTRIBUIÇÕES

O sistema criptográfico proposto se mostrou eficaz e simples de usar. A análise da decifragem mostrou que o sistema é seguro contra ataques de força bruta. Também foi aplicado um teste em uma sequência gerada pelo método usado no sistema e concluiu-se que ela poderia ser considerada aleatória satisfazendo os requisitos que o sistema necessita.

Foi analisado ainda que, para garantir o melhor uso do sistema, deve-se usar uma imagem base com predominância de pixels pretos e como mensagem uma imagem com texto fino e de tamanho médio (sugestão: Courier New 16), ocupando a maior área possível da imagem. Além disso, deve-se dividir o texto a ser enviado em várias partes e transmiti-las separadamente, para dificultar a recuperação do texto completo em uma interceptação.

9.2 TRABALHOS FUTUROS

Para dar continuidade a este trabalho, seria interessante construir um aplicativo para troca de mensagens entre sistemas móveis, bem como pesquisar outras aplicações do sistema para atividades do dia-a-dia.

Outro ponto interessante a ser estudado é o problema da distribuição da chave. Para fazer a decifragem são enviados três parâmetros ao destinatário: a imagem base, a chave secreta e a imagem de verificação. Analisamos que com dois desses parâmetros não é possível descobrir o conteúdo da mensagem, porém é importante garantir que um terceiro não consiga obter os três parâmetros ilegalmente.

Além disso, um ramo de pesquisa que vem ganhando força na criptografia visual é o estudo de fraudes. Seria interessante verificar como isso se aplica ao sistema proposto.

Por fim, para aprofundar o estudo sobre o assunto, pode-se testar outras técnicas de esteganografia aplicadas a esse trabalho e fazer um estudo comparativo entre elas.

10 REFERÊNCIAS BIBLIOGRÁFICAS

ANURADHA , S.; SAICHANDANA, B. **A New Visual Cryptography Scheme for Color Images.** International Journal of Engineering Science and Technology Vol. 2(6), 2010.

Camaleão. GitHub. Disponível em <https://github.com/carolinag/Camaleao>. Acesso em 18 de Março de 2013.

CHEN, K.; HUANG, S.; LEE, Y.; RAN-ZAN, W.; SHYU, S.. **Sharing multiple secrets in visual cryptography.** Pattern Recognition Journal. Ming Chuan University. Taiwan, março de 2007.

GitHub. Disponível em <https://github.com/>. Acesso em 18 de Março de 2013.

GOKUL, M.; SHRIRAM, K.; UMESHBAB**U**, R.; DEEPAK, K.. **Hybrid Steganography using Visual Cryptography and LSB Encryption Method.** International Journal of Computer Applications. Volume 59– No.14. .Dezembro de 2012.

DOMINGUES, F.B.. **Prevenção contra fraude em criptografia visual com base na autenticação de múltiplas imagens em transparências circulares.** Instituto Militar de Engenharia. Rio de Janeiro, fevereiro de 2013.

HWANG, R. **A digital image copyright protection scheme based on visual cryptography**. Tamkang Journal of Science and Engineering, 3(2):97-106. 2000.

Image Class. Disponível em <http://msdn.microsoft.com/en-us/library/system.drawing.image.aspx>. Acesso em 18 de março de 2013.

KAFRI, O.; KEREN, E. **Encryption of pictures and shapes by random grids.** Optics Letters, 1987.

JITHESH, K.; KUMAR, A. **Multi layer information hiding -a blend of Steganography and visual cryptography.** Journal of Theoretical and Applied Information Technology, 2005.

NAOR, M.; SHAMIR, A.. **Visual cryptography. In Proceedings of Advances in Cryptology.** Eurocrypt 94, LNCS Vol. 950, pages 1-12. Springer-Verlag, 1994.

Random Class. Disponível em <http://msdn.microsoft.com/en-us/library/system.random.aspx>. Acesso em 18 de março de 2013.

RMagick. Disponível em < http://rmagick.rubyforge.org>. Acesso em 18 de março de 2013.

RUKHIN, A.; SOTO, J.; NECHVATAL, J.; SMID, M.; BARKER, E.; LEIGH, S.; LEVENSON, M.; VANGEL, M.; BANKS, D.; HECKERT, A.; DRAY, J.; VO, S.. **A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications.** NIST special publication 800-22. 1ª Revisão. 2010.

Subtractive Generator. Disponível em < http://rosettacode.org/wiki/Subtractive_generator>. Acesso em 23 de maio de 2013.

TORRES, R.H. **Relatório do documento: a statistical test suite for random and pseudorandom number generators for cryptographic applications publicado pelo NIST.** Instituto Militar de Engenharia. Rio de Janeiro, 2010.

Trello. Disponível em <https://trello.com/>. Acesso em 18 de março de 2013.

Visual Studio Express. Microsoft. Disponível em <http://www.microsoft.com/visualstudio/eng/products/visual-studio-2010-express>. Acesso em 18 de março de 2013.

11 APÊNDICES

11.1 APÊNDICE 1: IMPLEMENTAÇÃO INICIAL

Este apêndice contém as classes *NaorShamirCypher* e *NaorShamirDecypher*, que implementam o método de Naor e Shamir *2 out of 2*. Os parâmetros de entrada de ambos os construtores são imagens *Bitmap* com o caminho da imagem a ser cifrada, na primeira classe, e das imagens a serem sobrepostas, na segunda classe.

11.1.1 CLASSE *NAORSHAMIRCYPHER.CS*

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing;

namespace Camaleao
{
   class NaorShamirCypher
   {
    private Bitmap Image { get; set; }
      public Bitmap Share1 { get; set; }
      public Bitmap Share2 { get; set; }
      private Random Rand { get; set; }

      public NaorShamirCypher(Bitmap image)
      {
         this.Image = image;
         this.Share1 = new Bitmap(image.Width * 2, image.Height * 2);
         this.Share2 = new Bitmap(image.Width * 2, image.Height * 2);
         Rand = new Random();
         GenerateShares();
      }

      private void GenerateShares()
      {
         for (int y = 0; y < Image.Height; y++)
         {
            for (int x = 0; x < Image.Width; x++)
            {
               Color imagePixel = Image.GetPixel(x, y);
               int randomInt = Rand.Next() % 2;
```

```
            if(imagePixel.R == 255)// Branco
            {
               if (randomInt == 0)
               {
                  Share1.SetPixel(2*x, 2*y, Color.Black);
                  Share1.SetPixel(2*x+1, 2*y, Color.White);
                  Share1.SetPixel(2*x, 2*y+1, Color.White);
                  Share1.SetPixel(2*x+1, 2*y+1, Color.Black);

                  Share2.SetPixel(2*x, 2*y, Color.Black);
                  Share2.SetPixel(2*x+1, 2*y, Color.White);
                  Share2.SetPixel(2*x, 2*y+1, Color.White);
                  Share2.SetPixel(2*x+1, 2*y+1, Color.Black);
               }
               else
               {
                  Share1.SetPixel(2*x, 2*y, Color.White);
                  Share1.SetPixel(2*x+1, 2*y, Color.Black);
                  Share1.SetPixel(2*x, 2*y+1, Color.Black);
                  Share1.SetPixel(2*x+1, 2*y+1, Color.White);

                  Share2.SetPixel(2*x, 2*y, Color.White);
                  Share2.SetPixel(2*x+1, 2*y, Color.Black);
                  Share2.SetPixel(2*x, 2*y+1, Color.Black);
                  Share2.SetPixel(2*x+1, 2*y+1, Color.White);
               }
            }
            else //Preto
            {
               if (randomInt == 0)
               {
                  Share1.SetPixel(2 * x, 2 * y, Color.Black);
                  Share1.SetPixel(2 * x + 1, 2 * y, Color.White);
                  Share1.SetPixel(2 * x, 2 * y + 1, Color.White);
                  Share1.SetPixel(2 * x + 1, 2 * y + 1, Color.Black);

                  Share2.SetPixel(2 * x, 2 * y, Color.White);
                  Share2.SetPixel(2 * x + 1, 2 * y, Color.Black);
                  Share2.SetPixel(2 * x, 2 * y + 1, Color.Black);
                  Share2.SetPixel(2 * x + 1, 2 * y + 1, Color.White);
               }
               else
               {
                  Share1.SetPixel(2 * x, 2 * y, Color.White);
                  Share1.SetPixel(2 * x + 1, 2 * y, Color.Black);
                  Share1.SetPixel(2 * x, 2 * y + 1, Color.Black);
                  Share1.SetPixel(2 * x + 1, 2 * y + 1, Color.White);

                  Share2.SetPixel(2 * x, 2 * y, Color.Black);
                  Share2.SetPixel(2 * x + 1, 2 * y, Color.White);
                  Share2.SetPixel(2 * x, 2 * y + 1, Color.White);
                  Share2.SetPixel(2 * x + 1, 2 * y + 1, Color.Black);
               }
            }
```

```
            }
        }
    }
  }
}
```

11.1.2 CLASSE *NAORSHAMIRDECYPHER.CS*

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing;

namespace Camaleao
{
   public class NaorShamirDecypher
   {
      public Bitmap RetrievedImage {get; set;}
      private Bitmap Share1 {get; set;}
      private Bitmap Share2 {get; set;}

      public NaorShamirDecypher(Bitmap share1, Bitmap share2)
      {
         this.Share1 = share1;
         this.Share2 = share2;
         this.RetrievedImage = new Bitmap(Share1.Width, Share1.Height);
         RetrieveImage();
      }

      private void RetrieveImage()
      {
          for (int y = 0; y <  RetrievedImage.Height; y++)
         {
            for (int x = 0; x < RetrievedImage.Width; x++)
            {
               Color pixel1 = Share1.GetPixel(x,y);
               Color pixel2 = Share2.GetPixel(x,y);

               int R = (pixel1.R + pixel2.R) == 255 ? 0 : ((pixel1.R + pixel2.R) > 255 ? 255 : (pixel1.R +
pixel2.R));

               int G = (pixel1.G + pixel2.G) == 255 ? 0 : ((pixel1.G + pixel2.G) > 255 ? 255 : (pixel1.G
+ pixel2.G));

               int B = (pixel1.B + pixel2.B) == 255 ? 0 : ((pixel1.B + pixel2.B) > 255 ? 255 : (pixel1.B +
pixel2.B));

               RetrievedImage.SetPixel(x, y, Color.FromArgb(255, R, G, B));
            }
         }
      }
   }
```

```
}
```

11.2 APÊNDICE 2: IMPLEMENTAÇÃO DO SISTEMA PROPOSTO

Este apêndice contém as classes *Cypher*, *Decypher* e *Utils*, que implementam o sistema proposto neste trabalho. Além disso, há também a classe *Program* que mostra como utilizar os métodos das classes acima.

11.2.1 CLASSE *CYPHER.CS*

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing;
using System.IO;
using System.Reflection;

namespace Camaleao
{
    public class Cypher
    {
        private Bitmap Message {get; set;}
        private Bitmap OriginalImage { get; set; }
        private Bitmap Share { get; set; }

        private int PrivateKey { get; set; }
        private Random Rand { get; set; }
        private List<int> R { get; set; }

        public Cypher(Bitmap message, Bitmap originalImage, int privateKey)
        {
            this.Message = message;
            this.OriginalImage = originalImage;
            this.Share = new Bitmap(Message.Width * 2, Message.Height);

            this.PrivateKey = privateKey;
            this.Rand = new Random(PrivateKey);
            R = new List<int>();
        }

        public Bitmap GenerateShare()
        {
            R = Utils.CreatePrivateKey(Rand, OriginalImage.Width *
OriginalImage.Height, Message.Width * Message.Height);

            for (int y = 0; y < Message.Height; y++)
            {
                for (int x = 0; x < Message.Width; x++)
                {
```

```
                    Color msgPixel = Message.GetPixel(x, y);

                    int yi= (int)R[y*Message.Width + x]/OriginalImage.Width;
                    int xi= R[y*Message.Width + x]%OriginalImage.Width;

                    Color imagePixel = OriginalImage.GetPixel(xi, yi);


                    if (msgPixel.Equals(Color.FromArgb(255,0,0,0)))
                   //pixel da mensagem é preto
                    {
                        //valido para imagens em tons de cinza (R = G = B)
                        if(imagePixel.R >= 80)
                        //se o bit mais a esquerda da cor do pixel selecionado = 1
                        {
                            Share.SetPixel(2 * x, y, Color.White);
                            Share.SetPixel(2 * x + 1, y, Color.Black);
                        }
                        else
                        {
                            Share.SetPixel(2 * x, y, Color.Black);
                            Share.SetPixel(2 * x + 1, y, Color.White);
                        }
                    }
                    else if (msgPixel.Equals(Color.FromArgb(255,255,255,255)))
                    //pixel da mensagem é branco
                    {
                        if (imagePixel.R >= 80)
                        //se o bit mais a esquerda da cor do pixel selecionado = 1
                        {
                            Share.SetPixel(2 * x, y, Color.Black);
                            Share.SetPixel(2 * x + 1, y, Color.White);
                        }
                        else
                        {
                            Share.SetPixel(2 * x, y, Color.White);
                            Share.SetPixel(2 * x + 1, y, Color.Black);
                        }
                    }

                }
            }

            TransformShare();

            return Share;
        }

        private void TransformShare()
        {
            Share = Utils.TransposeImage(Share);
            PermuteColumns();
            Share = Utils.TransposeImage(Share);
        }


        private void PermuteColumns()
        {
            Random random = new
Random((PrivateKey*OriginalImage.Width)%OriginalImage.Height);
```

```
            for (int i = 0; i < Share.Width; i++)
            {
                int column1 = random.Next(Share.Width);
                int column2 = random.Next(Share.Width);
                for (int j = 0; j < Share.Height; j++)
                {
                    Color Aux = Share.GetPixel(column1, j);
                    Share.SetPixel(column1, j, Share.GetPixel(column2, j));
                    Share.SetPixel(column2, j, Aux);
                }
            }
        }
    }
}
```

11.2.2 CLASSE *DECYPHER.CS*

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing;
using System.IO;
using System.Reflection;

namespace Camaleao
{
    public class Decypher
    {
        private Bitmap OriginalImage { get; set; }
        private Bitmap Share { get; set; }
        private Bitmap Message { get; set; }
        private int PrivateKey { get; set; }
        private Random Rand { get; set; }
        private List<int> R { get; set; }


        public Decypher(Bitmap originalImage, Bitmap share, int privateKey)
        {
            this.OriginalImage = originalImage;
            this.Share = share;
            this.Message = new Bitmap(Share.Width/2, Share.Height);

            this.PrivateKey = privateKey;
            this.Rand = new Random(PrivateKey);
            R = new List<int>();

        }

        public Bitmap GetMessage()
        {
            R = Utils.CreatePrivateKey(Rand, OriginalImage.Width *
OriginalImage.Height, Message.Width * Message.Height);

            TransformShareBack();

            for (int y = 0; y < Message.Height; y++)
            {
```

```
                for (int x = 0; x < Message.Width; x++)
                {
                    Color sharePixel1 = Share.GetPixel(2*x, y);
                    Color sharePixel2 = Share.GetPixel(2*x+1, y);

                    Color f1;
                    Color f2;

                    int yi = (int)R[y * Message.Width + x] / OriginalImage.Width;
                    int xi = R[y * Message.Width + x] % OriginalImage.Width;
                    Color imagePixel = OriginalImage.GetPixel(xi, yi);


                    if (imagePixel.Name.StartsWith("ff8") ||
                    imagePixel.Name.StartsWith("ff9") ||
                    imagePixel.Name.StartsWith("ffa") ||
                    imagePixel.Name.StartsWith("ffb") ||
                    imagePixel.Name.StartsWith("ffc") ||
                    imagePixel.Name.StartsWith("ffd") ||
                    imagePixel.Name.StartsWith("ffe") ||
                    imagePixel.Name.StartsWith("fff"))
                    {
                        f1 = Color.FromArgb(255,0,0,0);
                        f2 = Color.FromArgb(255,255,255);
                    }
                    else
                    {
                        f1 = Color.FromArgb(255, 255, 255);
                        f2 = Color.FromArgb(255, 0, 0, 0);
                    }

                    if ((f1.Name == sharePixel1.Name) && (f2.Name ==
sharePixel2.Name))
                    {
                        Message.SetPixel(x, y, Color.White);
                    }
                    else
                    {
                        Message.SetPixel(x, y, Color.Black);
                    }

                }
            }
            return Message;
        }


        private void TransformShareBack()
        {
            Share = Utils.TransposeImage(Share);
            PermuteColumnsBack();
            Share = Utils.TransposeImage(Share);
        }


        private void PermuteColumnsBack()
        {
            Random random = new Random((PrivateKey * OriginalImage.Width) %
OriginalImage.Height);
            List<int> r = Utils.CreatePrivateKey(random, Share.Width, 2*Share.Width);
```

```
            List<int> rInverso = new List<int>();

            for (int a = 1; a <= r.Count; a++)
            {
                rInverso.Add(r[r.Count - a]);
            }


            for (int i = 0; i < Share.Width; i++)
            {
                int column1 = rInverso[2*i];
                int column2 = rInverso[2*i+1];

                for (int j = 0; j < Share.Height; j++)
                {
                    Color Aux = Share.GetPixel(column1, j);
                    Share.SetPixel(column1, j, Share.GetPixel(column2, j));
                    Share.SetPixel(column2, j, Aux);
                }
            }
        }
    }
}
```

11.2.3 CLASSE *UTILS.CS*

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing;

namespace Camaleao
{
    public static class Utils
    {
        public static Bitmap TransposeImage(Bitmap image)
        {
            Bitmap imageT = new Bitmap(image.Height, image.Width);

            for (int y = 0; y < image.Height; y++)
            {
                for (int x = 0; x < image.Width; x++)
                {
                    imageT.SetPixel(y, x, image.GetPixel(x, y));
                }
            }
            return imageT;

        }

        public static List<int> CreatePrivateKey(Random random, int maxValue,int
keyLength)
        {
            List<int> r = new List<int>();
            for (int i = 0; i < keyLength; i++)
            {
                r.Add(random.Next(maxValue));
            }
```

```
            return r;
        }
    }
}
```

11.2.4 CLASSE *PROGRAM.CS*

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing;
using System.IO;
using System.Reflection;

namespace Camaleao
{
    class Program
    {
        public static Cypher cypher;
        public static Decypher decypher;

        public static NaorShamirCypher NScypher;
        public static NaorShamirDecypher NSDecypher;

        public static int key = 7;

        public static string AbsolutePath =  Path.GetDirectoryName
(Assembly.GetExecutingAssembly().Location);

        static void Main(string[] args)
        {
            // cifragem: geração da imagem de verificação (share)
            Bitmap originalImage = new Bitmap(Path.Combine (AbsolutePath,
"../../Images/Chameleon4.bmp"));
            Bitmap message = new Bitmap(Path.Combine(AbsolutePath,
"../../Images/message.bmp"));

            cypher = new Cypher(message, originalImage, key);
            Bitmap share = cypher.GenerateShare();
            share.Save(Path.Combine(AbsolutePath,
"../../Images/shareChameleon4.bmp"));

            // decifragem: receptor pega a imagem de verificação e encontra a mensagem
            Bitmap receivedShare = new Bitmap(Path.Combine(AbsolutePath,
"../../Images/shareChameleon4.bmp"));

            decypher = new Decypher(originalImage, share, key);
            Bitmap retrievedMessage = decypher.GetMessage();
            retrievedMessage.Save(Path.Combine(AbsolutePath,
"../../Images/retrievedMessage.bmp"));
         }
    }
}
```